团队交付说明

项目：疾病症状共现模式挖掘与关联规则分析

一、总体思路

我们采用**Apriori算法**对疾病症状数据集进行频繁项集挖掘与关联规则分析，目标是发现症状之间的共现模式。

具体步骤：

1. **数据清洗与标准化**：清理症状名称，处理同义词，统一格式
2. **转换为事务格式**：将每个疾病的症状集合视为一个事务（transaction）
3. **频繁项集挖掘**：使用Apriori算法找出频繁出现的症状组合
4. **关联规则生成**：从频繁项集中提取有意义的关联规则
5. **敏感性分析**：测试不同支持度阈值对结果的影响

最终目的是找到：

- 哪些症状组合在疾病中频繁共同出现
- 症状之间存在哪些强关联关系

二、数据预处理部分

🔵 1. 数据加载与基本统计

```
# 加载数据集
df = pd.read_csv('dataset.csv')

# 数据集结构
# - 4920行（疾病实例）
# - 18列（1列疾病名 + 17列症状）
# - 41种不同疾病
```

数据特点：

- 每行代表一个疾病实例
- 症状列有大量缺失值（后期症状列缺失较多）
- 平均每个疾病有7.43个症状

🔵 2. 症状清洗与标准化

```
def clean_symptom(symptom):
    """
    清洗和标准化症状名称
    - 转换为小写
    - 删除首尾空格
    - 替换下划线为空格
    """
    if pd.isna(symptom) or symptom == '':
        return None

    symptom = str(symptom).lower().strip()
    symptom = symptom.replace('_', ' ')
    return symptom if symptom else None
```

同义词映射：

```
symptom_synonyms = {
    'pyrexia': 'fever',
    'high temperature': 'fever',
    'headache': 'head ache',
    'stomach ache': 'abdominal pain',
    'difficulty breathing': 'breathlessness',
    'throwing up': 'vomiting',
    'loose motions': 'diarrhoea',
    'tiredness': 'fatigue',
    # ... 更多同义词映射
}
```

作用：

- 统一症状命名规范
- 合并同义症状，提高数据质量
- 清理数据中的格式不一致问题

🔵 3. 转换为事务格式

将数据转换为Apriori算法所需的事务格式：

```
def prepare_transactions(df):
    """
    每个疾病实例 → 一个事务（transaction）
    事务内容 = 该疾病的所有症状集合
    """
    transactions = []
    disease_names = []

    for idx, row in df.iterrows():
        symptoms = set()

        # 收集该疾病的所有症状
        for col in symptom_columns:
            symptom = clean_symptom(row[col])
            if symptom:
                symptom = normalize_symptom(symptom)
                symptoms.add(symptom)

        if symptoms:
            transactions.append(symptoms)
            disease_names.append(row['Disease'])

    return transactions, disease_names
```

输出示例：

```
Transaction 1: {'itching', 'skin rash', 'nodal skin eruptions',
'dischromic patches'}
Transaction 2: {'fatigue', 'vomiting', 'high fever', 'loss of appetite'}
...
```

三、Apriori算法实现

🔵 1. 算法核心参数

```
MIN_SUPPORT = 0.05       # 最小支持度 5%
MIN_CONFIDENCE = 0.6     # 最小置信度 60%
```

参数说明：

- **支持度（Support）**：项集在所有事务中出现的频率
  - `Support(X) = 包含X的事务数 / 总事务数`
- **置信度（Confidence）**：规则 A→B 的可靠程度
  - `Confidence(A→B) = Support(A∪B) / Support(A)`
- **提升度（Lift）**：规则的实际效果与独立情况的比值
  - `Lift(A→B) = Confidence(A→B) / Support(B)`
  - Lift > 1：正相关；Lift = 1：独立；Lift < 1：负相关

🔵 2. Apriori算法流程

步骤1：找出频繁1-项集

```
def _get_frequent_1_itemsets(self):
    """扫描所有事务，统计单个症状的出现频率"""
    item_counts = defaultdict(int)

    for transaction in self.transactions:
        for item in transaction:
            item_counts[frozenset([item])] += 1

    # 过滤掉支持度小于阈值的项
    min_count = self.min_support * len(self.transactions)
    frequent_items = {
        itemset: count / len(self.transactions)
        for itemset, count in item_counts.items()
        if count >= min_count
    }

    return frequent_items
```

步骤2：候选集生成（自连接）

```
def _generate_candidates(self, frequent_itemsets_k):
    """
```

```
    从k-项集生成(k+1)-候选项集
    方法：自连接 - 合并两个k-项集
    """
    candidates = set()
    itemsets_list = list(frequent_itemsets_k.keys())

    for i in range(len(itemsets_list)):
        for j in range(i + 1, len(itemsets_list)):
            union = itemsets_list[i] | itemsets_list[j]

            # 如果合并后大小为k+1，则是有效候选
            if len(union) == len(itemsets_list[i]) + 1:
                candidates.add(union)

    return candidates
```

步骤3：候选集剪枝

```
def _prune_candidates(self, candidates, frequent_itemsets_k):
    """
    剪枝：移除子集不频繁的候选项集
    Apriori性质：频繁项集的所有子集也必须是频繁的
    """
    pruned = set()

    for candidate in candidates:
        # 检查所有k-子集是否都频繁
        subsets = [frozenset(s) for s in combinations(candidate, len(canc

        if all(subset in frequent_itemsets_k for subset in subsets):
            pruned.add(candidate)

    return pruned
```

步骤4：迭代挖掘频繁项集

```
def _find_frequent_itemsets(self):
    """
    迭代生成所有频繁项集
    从1-项集开始，逐步生成更大的频繁项集
    """
```

```
    all_frequent_itemsets = {}

    # 找出频繁1-项集
    frequent_1 = self._get_frequent_1_itemsets()
    all_frequent_itemsets[1] = frequent_1

    k = 1
    while True:
        # 生成候选项集
        candidates = self._generate_candidates(all_frequent_itemsets[k])
        if not candidates:
            break

        # 剪枝
        candidates = self._prune_candidates(candidates, all_frequent_item
        if not candidates:
            break

        # 计算候选项集的支持度
        frequent_k_plus_1 = {}
        for candidate in candidates:
            support = self._calculate_support(candidate)
            if support >= self.min_support:
                frequent_k_plus_1[candidate] = support

        if not frequent_k_plus_1:
            break

        k += 1
        all_frequent_itemsets[k] = frequent_k_plus_1

    return all_frequent_itemsets
```

🔵 3. 关联规则生成

```
def _generate_association_rules(self):
    """
    从频繁项集生成关联规则
    规则形式：A → B
    """
    rules = []
```

```
        # 从2-项集开始生成规则
        for k in range(2, max(self.frequent_itemsets.keys()) + 1):
            for itemset, support in self.frequent_itemsets[k].items():
                # 生成所有非空真子集作为前件
                for i in range(1, len(itemset)):
                    for antecedent in combinations(itemset, i):
                        antecedent = frozenset(antecedent)
                        consequent = itemset - antecedent

                        # 计算置信度
                        antecedent_support = self._get_support(antecedent)
                        if antecedent_support > 0:
                            confidence = support / antecedent_support

                            if confidence >= self.min_confidence:
                                # 计算提升度
                                consequent_support = self._get_support(consec
                                lift = confidence / consequent_support if cor

                                rules.append({
                                    'antecedent': antecedent,
                                    'consequent': consequent,
                                    'support': support,
                                    'confidence': confidence,
                                    'lift': lift
                                })

        return rules
```

四、运行算法与结果分析

🔵 1. 执行挖掘

```
# 创建并训练模型
apriori = AprioriAlgorithm(min_support=MIN_SUPPORT, min_confidence=MIN_CO
apriori.fit(transactions)

# 获取结果
frequent_itemsets_df = apriori.get_frequent_itemsets_df()
association_rules_df = apriori.get_association_rules_df()
```

🔵 2. 频繁项集统计

发现的频繁项集数量（支持度≥5%）：

项集大小	数量
1-项集	31
2-项集	104
3-项集	153
4-项集	143
5-项集	86
6-项集	36
7-项集	9
总计	**562**

🔵 3. Top 5 最频繁症状（1-项集）

症状	支持度	出现次数
fatigue（疲劳）	0.393	1932
vomiting（呕吐）	0.389	1914
high fever（高烧）	0.277	1362
loss of appetite（食欲不振）	0.234	1152
nausea（恶心）	0.233	1146

解读：疲劳和呕吐是最常见的症状，在近40%的疾病中出现。

🔵 4. Top 5 最频繁症状组合（2-项集）

症状组合	支持度
nausea + vomiting（恶心+呕吐）	0.199
fatigue + high fever（疲劳+高烧）	0.199
abdominal pain + vomiting（腹痛+呕吐）	0.178

症状组合	支持度
loss of appetite + yellowing of eyes（食欲不振+眼睛发黄）	0.160
fatigue + loss of appetite（疲劳+食欲不振）	0.157

解读：这些症状组合具有明显的医学意义，例如恶心和呕吐常常一起出现。

🔵 5. 关联规则分析

发现的关联规则数量：5458条

Top 5 最强关联规则（按Lift排序）：

前件	后件	支持度	置信度	提升度
phlegm	chest pain, malaise	0.070	0.966	13.659
chest pain, malaise	phlegm	0.070	0.983	13.659
phlegm	chest pain, fatigue	0.068	0.949	13.654
phlegm	chest pain, chills	0.068	0.949	13.654
chest pain, chills	phlegm	0.068	0.982	13.654

关键发现：

- **Lift > 13**：表示这些症状组合的共现频率远高于随机情况
- **置信度 > 96%**：如果出现前件症状，则后件症状有极高概率同时出现
- **临床意义**：痰液（phlegm）与胸痛、不适等症状高度相关，可能指示呼吸系统疾病

五、敏感性分析

测试不同最小支持度阈值对结果的影响：

🔵 测试结果

最小支持度	总频繁项集数	1-项集	2-项集	3-项集	关联规则数
0.02	2426	55	437	842	138,922
0.05	562	31	104	153	5,458
0.10	140	16	34	41	382

最小支持度	总频繁项集数	1-项集	2-项集	3-项集	关联规则数
0.15	48	11	16	13	44
0.20	20	7	8	4	8

🔵 敏感性分析图表说明

1. **总频繁项集数 vs 支持度**：
   - 支持度降低，频繁项集数量呈指数级增长
   - 支持度过低会产生大量噪声模式
2. **关联规则数 vs 支持度**：
   - 支持度从0.02到0.20，规则数量从138,922降至8
   - 需要在规则数量和规则质量之间权衡
3. **推荐参数**：
   - **支持度 = 0.05**：平衡了模式数量和模式质量
   - 既能发现足够多的模式，又避免了过多噪声

六、结果解释（核心发现）

🔵 1. 数据集概况

- **总疾病实例数**：4920
- **唯一症状数**：130
- **平均每疾病症状数**：7.43

🔵 2. 重要医学发现

消化系统症状组合

症状组合	含义
恶心 + 呕吐	消化系统紊乱的典型症状
腹痛 + 呕吐	可能指示急性胃肠炎或其他消化问题
食欲不振 + 眼睛发黄	可能提示肝胆系统疾病

全身性症状组合

症状组合	含义
疲劳 + 高烧	常见于感染性疾病
疲劳 + 食欲不振	多种慢性疾病的共同表现

呼吸系统症状组合

症状组合	含义
痰液 + 胸痛 + 不适	强烈提示呼吸系统疾病（如肺炎、支气管炎）
痰液 + 胸痛 + 寒战	可能是感染性呼吸道疾病

🔵 3. 强关联规则的医学意义

规则示例：`phlegm → chest pain, malaise` (Lift = 13.659)

解读：

- 如果患者有痰液症状，有96.6%的概率会伴随胸痛和不适
- 这个组合的出现频率是随机情况的13.6倍
- **临床应用**：当患者主诉有痰时，医生应关注是否有胸痛和全身不适

规则示例：`fatigue, high fever → vomiting` (假设)

解读：

- 疲劳和高烧的患者很可能会出现呕吐
- 提示医生需要关注患者的水分摄入和电解质平衡

七、注意事项

⚠️ 算法使用注意事项

1. **支持度阈值设置**：
    - 支持度过高（>0.2）：会遗漏重要但较少见的症状组合
    - 支持度过低（<0.02）：会产生大量无意义的模式
    - **推荐值**：0.05 - 0.1，根据具体需求调整

2. **置信度阈值设置**：
    - 置信度过低（<0.5）：关联规则可能不可靠
    - 置信度过高（>0.9）：虽然可靠但可能太严格
    - **推荐值**：0.6 - 0.8
3. **Lift值解读**：
    - Lift ≈ 1：前件和后件独立，无关联
    - Lift > 1：正相关，前件出现会增加后件出现的概率
    - Lift < 1：负相关，前件出现会降低后件出现的概率
    - **关注重点**：Lift > 2的规则通常有实际意义

⚠️ 数据质量注意事项

1. **症状同义词处理**：
    - 已在代码中处理了常见同义词
    - 如果数据集更新，需要检查是否有新的同义词需要映射
2. **缺失值处理**：
    - 症状列后期缺失较多是正常的（并非所有疾病都有17个症状）
    - 代码已自动过滤空值
3. **数据来源**：
    - 数据集来自Kaggle：Disease Symptom Description Dataset
    - 请确保使用最新版本的数据集

⚠️ 结果使用注意事项

1. **医学验证**：
    - 所有发现的症状模式和关联规则应由医学专业人士进行验证
    - 数据挖掘结果不能直接用于临床诊断，仅供参考
2. **因果关系**：
    - 关联规则表示的是**共现关系**，不是**因果关系**
    - 例如：A → B 不意味着A导致B，只是它们经常一起出现
3. **样本偏差**：
    - 数据集中疾病分布可能不均匀

- 某些罕见病可能样本不足，影响结果的普遍性

⚠️ 运行环境要求

```
# 必需的Python库
pandas >= 1.3.0
numpy >= 1.21.0
matplotlib >= 3.4.0
seaborn >= 0.11.0
```

运行时间：

- 小数据集（<5000条）：1-2分钟
- 中等数据集（5000-10000条）：3-5分钟
- 支持度越低，运行时间越长（指数级增长）

⚠️ 输出文件说明

运行完成后会生成以下CSV文件：

1. `task1_frequent_itemsets.csv`：
    - 包含所有频繁项集及其支持度
    - 可用于进一步分析或可视化
2. `task1_association_rules.csv`：
    - 包含所有关联规则及其支持度、置信度、提升度
    - 按提升度降序排列
3. `task1_sensitivity_analysis.csv`：
    - 不同支持度阈值下的结果统计
    - 用于选择最优参数

八、代码使用指南

🔵 快速开始

1. **准备数据**：

```
# 确保dataset.csv在项目根目录
ls dataset.csv
```

2. **运行Notebook**：

```
jupyter notebook task1.ipynb
```

3. **执行所有单元格**：
   - 点击 `Kernel` → `Restart & Run All`
   - 等待2-3分钟完成
4. **查看结果**：
   - 在Notebook中查看可视化图表
   - 检查生成的CSV文件

🔵 自定义参数

如果要调整参数，修改以下代码块：

```
# 在 Cell 17 中修改
MIN_SUPPORT = 0.05       # 修改为你需要的支持度
MIN_CONFIDENCE = 0.6     # 修改为你需要的置信度
```

然后重新运行后续单元格。

🔵 添加新的同义词

如果发现数据中有新的同义词需要处理：

```
# 在 Cell 9 中添加
symptom_synonyms = {
    # 已有映射...
    'your_synonym': 'standard_name',   # 添加新的映射
}
```

九、总结与展望

✅ 已完成的工作

1. ✅ 完整实现了Apriori算法（自主编写）
2. ✅ 对症状数据进行了全面的预处理和清洗
3. ✅ 发现了562个频繁症状组合模式
4. ✅ 生成了5458条关联规则
5. ✅ 进行了敏感性分析，验证了参数选择的合理性
6. ✅ 提供了详细的可视化分析

🎯 核心价值

1. **辅助诊断**：帮助医生识别症状模式，进行初步诊断
2. **疾病预测**：基于已知症状预测可能的伴随症状
3. **知识发现**：从数据中发现症状之间的隐藏关联
4. **参数优化**：通过敏感性分析找到最优挖掘参数

🚀 未来改进方向

1. **算法扩展**：
   - 实现FP-Growth算法，与Apriori对比效率
   - 尝试序列模式挖掘（如Task2的PrefixSpan）
2. **特征增强**：
   - 考虑症状的严重程度（轻/中/重）
   - 引入症状的时序信息（先后顺序）
   - 结合患者人口统计学特征（年龄、性别）
3. **医学验证**：
   - 邀请医学专家评审发现的规则
   - 与医学文献对比验证
   - 进行临床试验验证
4. **可视化增强**：
   - 构建症状关联网络图
   - 开发交互式可视化界面

- 按疾病类别进行分组可视化

5. **应用扩展**：
   - 开发症状查询系统
   - 构建疾病预测模型
   - 集成到临床决策支持系统

十、参考资料

📚 算法参考

- **Apriori算法原理**：
  - Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. *Proceedings of VLDB*, 487-499.
- **关联规则度量指标**：
  - Support, Confidence, Lift的定义和计算方法

📊 数据来源

- **Dataset**: Disease Symptom Description Dataset
- **Source**: Kaggle
- **License**: Public Domain

💡 相关链接

- **项目GitHub**：SC4020DiseaseMining
- **运行指南**：参见 `TASK1_运行指南.md`

End of Document

最后更新时间：2025年11月1日

项目成员：[团队成员名单]

联系方式：[联系邮箱]

声明：本项目仅用于学术研究和教学目的。所有数据挖掘结果需经医学专业人士验证后方可用于实际应用。不得将本项目结果直接用于临床诊断或治疗决策。